Apple Assembly Line

$1.50

Volume 3 -- Issue 9 June, 1983

In This Issue...

Spiral Screen Clear 2
Breaking and Entering 9
Binary to Decimal Conversion 11
The Tiniest Motherboard 15
Replacing INIT Can Be Dangerous 16
Reformatting a Lot of Text 19
Track Balls 24
Ampersand Monitor Caller 30

A New Look?

The cover and a few of the inside pages of this issue look a little different, don't they? Well, our offices were burglarized last night, and the Spinwriter we use for newsletter printing was damaged. The thieves also made off with three complete computer systems and a load of software. They even got the Track Ball I used for the article in this issue! See page 9 for more details.

65C02 Note

Don Lancaster just called to report that he has gotten his hands on samples of the GTE G65SC02 processor. It does drop right into his Apple //e, and runs just fine! Note that this is the GTE version, which does not have the instructions to set, reset, and test single bits. Those are in the Rockwell chip, which still hasn't shown up. We'll keep passing on whatever we hear.

Spiral Screen Clear................................Roger Keating

modified and documented by Bob Sander-Cederlof

[Roger is the author of numerous excellent war games for the Apple, some published by Strategic Simulations. Titles include Southern Command, Operation Apocalypse, Germany 1985, and Rapid Deployment Force. He is also director for Australasia of the International Apple Core. A previous version of this program was published in the newsletter of one of the Australian Apple user groups.]

The following program demonstrates an innovative method for clearing the Apple text or lo-res graphics screen. Rather than just switching to instant blackness, it makes the process visually interesting.

The entire screen is viewed as one long coiled line. For 55 seconds this line unwinds on the screen, with blanks being fed into the center and visible characters shifting out at the bottom left corner. Here is a simplified diagram, on a 10-column 6-line screen:

```
  0   1   2   3   4   5   6   7   8   9
-----------------------------------------
| V | < | < | < | < | < | < | < | < | < | 0
--  ---------------------------------  --
| V | V | < | < | < | < | < | < | < | ^ | 1
--  ---  -------------------------  ---  --
| V | V | V | < | < | < | < | < | ^ | ^ | 2
--  ---  ---  ---------------------  ---  ---  --
| V | V | V | > | > | > | > | ^ | ^ | ^ | 3
--  ---  ---  -------------------------  ---  --
| V | V | > | > | > | > | > | > | ^ | ^ | 4
--  ---  ---------------------------------  --
| X | > | > | > | > | > | > | > | > | ^ | 5
--  -------------------------------------
  0   1   2   3   4   5   6   7   8   9
```

There are a total of 60 characters. By calling 60 times a routine which rotates the spiral, and inserting a blank at line 3 column 3 after each rotation, I can blank out the entire 60 characters. The <, >, V, and ^ symbols show the direction each character cell will move during the rotation. The "X" in line 5 column 0 is shifted out to never-never land. (In the old days, we said it went into the "bit bucket".)

Well, that is the general idea. But I did it for the whole 24x40 screen, a little too much to show in this small space.

Here is a description of the highest level of the spiral clear program, in a language a little like BASIC:

```
     FOR VCNT = 1 TO 24
         FOR HCNT = 1 TO 40
             ROTATE SCREEN
             STORE BLANK AT TRAIL'S END
         NEXT HCNT
     NEXT VCNT
```

S-C Macro Assembler (the best there is!)	$80.00
S-C Macro Assembler Version 1.1 Update	$12.50

S-C Cross Reference Utility	$20.00
S-C Cross Reference Utility with Complete Source Code	$50.00

S-C Word Processor As is, with fully commented source code. Needs S-C Macro Assembler.	$50.00
Applesoft Source Code on Disk Very heavily commented. Requires Applesoft and S-C Assembler.	$50.00
ES-CAPE: Extended S-C Applesoft Program Editor	$60.00

AAL Quarterly Disks	each $15.00

Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	

Double Precision Floating Point for Applesoft Provides 21-digit precision for Applesoft programs. Includes sample Applesoft subroutines for standard math functions.	$50.00

FLASH! Integer BASIC Compiler (Laumer Research)	$79.00
Source Code for FLASH! Runtime Package	$39.00
Full Screen Editor for S-C Macro Assembler (Laumer Research)	$49.00

The Visible Computer: 6502 (Software Masters)	(reg. $50.00)	$45.00
Super Disk Copy III (Sensible Software)	(reg. $30.00)	$27.00
Amper-Magic (Anthro-Digital)	(reg. $75.00)	$67.50
Amper-Magic Volume 2 (Anthro-Digital)	(reg. $35.00)	$30.00
Quick-Trace (Anthro-Digital)	(reg. $50.00)	$45.00
DISASM Dis-Assembler (RAK-Ware)		$30.00

Blank Diskettes (with hub rings)	package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks	$36.00
Vinyl disk pages, 6"x8.5", hold one disk each	10 for $6.00
Reload your own NEC PC-8023 ribbon cartridges	each ribbon $5.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges	each $2.50
Diskette Mailing Protectors	10-99: 40 cents each 100 or more: 25 cents each
ZIF Game Socket Extender	$20.00
Ashby Shift-Key Mod	$15.00
Lower-Case Display Encoder ROM Only Revision level 7 or later Apples.	$25.00

STB-80 80-column Display Board (STB Systems)	($249.00)	$225.00
STB-128 128K RAM Card (STB Systems)	($399.00)	$350.00

Grappler+ Printer Interface (Orange Micro)	($175.00)	$150.00
Bufferboard 16K Buffer for Grappler (Orange Micro)	($175.00)	$150.00
Buffered Grappler+ NEW!! Interface and 16K Buffer	($239.00)	$200.00

Books, Books, Books..........compare our discount prices!

"The Apple][Circuit Description", Gayler	($22.95)	$21.00
"Enhancing Your Apple II, vol. 1", Lancaster	($17.95)	$17.00
"Incredible Secret Money Machine", Lancaster	($7.95)	$7.50
"Micro Cookbook, vol. 1", Lancaster	($15.95)	$15.00
"Beneath Apple DOS", Worth & Lechner	($19.95)	$18.00
"Bag of Tricks", Worth & Lechner, with diskette	($39.95)	$36.00
"Apple Graphics & Arcade Game Design", Stanton	($19.95)	$18.00
"Assembly Lines: The Book", Roger Wagner	($19.95)	$18.00
"What's Where in the Apple", Second Edition	($24.95)	$23.00
"What's Where Guide" (updates first edition)	($9.95)	$9.00
"6502 Assembly Language Programming", Leventhal	($16.99)	$16.00
"6502 Subroutines", Leventhal	($15.99)	$15.00

Add $1 per book for US postage. Foreign orders add postage needed.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We take Master Charge, VISA and American Express ***

The use of two loops is not necessary, because the loop variables are not used at all inside the loops. We could just as well write it like this:

```
     FOR CNT = 1 TO 960
          ROTATE SCREEN
          STORE BLANK
     NEXT CNT
```

However, thinking ahead to the assembly language implementation, we know that counters are easier to manage if they have values less than 256. Therefore I like the first version better. Furthermore, counters in assembly language are frequently easier to work with if they count backwards. I ended up with the code in lines 1130-1270 of the program.

The value "$628+12" in line 1210 happens to be the memory address of the inner end of the spiral. I figured it out on paper, tried it, corrected my figuring, and tried it again.

You can add some cute wrinkles here and there. Like putting a "pause if key pressed" routine right after calls to ROTATE.SCREEN. Like storing something besides a blank. Like feeding the character from the bottom left corner into line 12 column 12, so that the original text is restored. Like varying the character stored during the 960 rotations. Like using a color value in lo-res graphics mode. I tried a lot of these, and it is amazing how much you can learn this way.

The rotation process involves four separate steps: sliding a column on the left side down, slipping an upper row to the left, shoving a column on the right side up, and scooting a lower row to the right. Appealing once again to a higher level language, it might look like this:

```
     XL=0 : XR=39
     YT=0 : YB=23
     WHILE YT<YB
          SLIDE COLUMN XL DOWN
          SLIP ROW YT LEFT
          SHOVE COLUMN XR UP
          SCOOT ROW YB RIGHT
          ADJUST XL, XR, YT, YB
     LOOP
```

It turns out it is not quite that simple, but almost. The assembly language is in lines 1300-1530.

The initial call to DOWN at line 1390 moves the leftmost column of characters down, ignoring the bottom left cell. If that cell were not ignored, it would slide to some undetermined place in memory, not necessarily even on the screen. Where it goes depends on what method is used to compute screen addresses from line number. Anyway, it would try to go to a 25th line, which does not exist.

It turns out that if BASCALC in the monitor ROM is used, the character is moved into $478, which is not part of the screen. It is usually a safe location, but some peripheral boards or DOS might use it. If you don't care about saving $478, or if you use a different method which leads to a completely safe address for the 25th line, you could omit lines 1390-1400.

I still need to show you the routines DOWN, LEFT, UP, and RIGHT. Here they are in pseudo-code:

```
DOWN:  FOR Y = YB-1 TO YT STEP -1
           S(XL,Y+1) = S(XL,Y)
       NEXT Y

LEFT:  FOR X = XL+1 TO XR
           S(X-1,YT) = S(X,YT)
       NEXT X

UP:    FOR Y = YT+1 TO YB
           S(XR,Y-1) = S(XR,Y)
       NEXT Y

RIGHT: FOR X = XR-1 TO XL STEP -1
           S(X+1,YB) = S(X,YB)
       NEXT X
```

The assembly language is in lines 1550-2240.

Each of these routines needs a function to compute the base address of a line on the screen. I called the subroutine MAKE.BASE, and implemented it in two different ways. The easiest way is to call on the subroutine BASCALC at $FBC1 in the monitor ROM.

BASCALC accepts the line number (0-23) in the A-register, computes the base address by a series of shift and masking operations, and puts the beginning address of the line into $28 and $29. No other registers are used, so it is a nice subroutine to have available. (Its only problem is that it takes a full 40 microseconds to do the job.) Since I call MAKE.BASE with the line number in the X-register, it can be written like lines 2310-2320.

The faster way to get a base address loaded is to use a table of addresses. I put the high byte of each base address in a 24-byte table, and the low byte in another. The line number in the X-register indexes this table. MAKE.BASE is in lnes 2340-2380, and the table in lines 2400-2450.

The table lookup is about twice as fast as BASCALC's 40 microseconds. Since MAKE.BASE is called once each for LEFT and RIGHT, and twice for each character moved for UP and DOWN, and the whole set is called 960 times, the overall effect is large. Using BASCALC the total time to clear the screen is 55 seconds; with the table lookup it is 40 seconds. On the other hand, the table and the code to read it take up 59 bytes, while the call to BASCALC takes only 4 bytes.

```
               1000 *--------------------------------
               1010 *      SPIRAL CLEAR BY ROGER KEATING
               1020 *--------------------------------
0000-          1030 HCNT   .EQ $00
0001-          1040 VCNT   .EQ $01
0002-          1050 XL     .EQ $02
0003-          1060 XR     .EQ $03
0004-          1070 YT     .EQ $04
0005-          1080 YB     .EQ $05
0028-          1090 BASE   .EQ $28
               1100 *--------------------------------
               1110        .OR $800
               1120 *--------------------------------
               1130 SPIRAL.CLEAR
0800- A9 18    1140        LDA #24      FOR 24 LINES
0802- 85 01    1150        STA VCNT
0804- A9 28    1160 .1     LDA #40      FOR 40 COLUMNS
0806- 85 00    1170        STA HCNT
               1180
0808- 20 19 08 1190 .2     JSR ROTATE.SCREEN
080B- A9 A0    1200        LDA #$A0     STORE BLANK IN
080D- 8D 34 06 1210        STA $628+12  MIDDLE OF SCREEN
               1220
0810- C6 00    1230        DEC HCNT     NEXT HCNT
0812- D0 F4    1240        BNE .2
0814- C6 01    1250        DEC VCNT     NEXT VCNT
0816- D0 EC    1260        BNE .1
0818- 60       1270        RTS          FINISHED!
               1280
               1290 *--------------------------------
               1300 ROTATE.SCREEN
0819- A9 00    1310        LDA #0
081B- 85 02    1320        STA XL       LEFT END
081D- 85 04    1330        STA YT       TOP
081F- A9 17    1340        LDA #23
0821- 85 05    1350        STA YB       BOTTOM
0823- A9 27    1360        LDA #39
0825- 85 03    1370        STA XR       RIGHT END
               1380
0827- 20 47 08 1390        JSR DOWN     START LEFT SIDE
082A- F0 05    1400        BEQ .2       ...ALWAYS
               1410
082C- 20 47 08 1420 .1     JSR DOWN     SLIDE LEFT SIDE DOWN
082F- C6 05    1430        DEC YB       MOVE BOTTOM UP
0831- 20 77 08 1440 .2     JSR LEFT     SLIP TOP LINE LEFT
0834- E6 02    1450        INC XL       MOVE LEFT EDGE IN
0836- 20 5F 08 1460        JSR UP       SHOVE RIGHT SIDE UP
0839- E6 04    1470        INC YT       MOVE TOP DOWN
083B- 20 8A 08 1480        JSR RIGHT    SCOOT BOTTOM LINE RIGHT
083E- C6 03    1490        DEC XR       MOVE RIGHT EDGE IN
0840- A5 04    1500        LDA YT
0842- C5 05    1510        CMP YB
0844- 90 E6    1520        BCC .1       IF YT<YB, BRANCH
0846- 60       1530        RTS          FINISHED
               1540
               1550 *--------------------------------
               1560 *      MOVE LEFT SIDE DOWN
               1570 *      FOR Y=YB-1 TO YT STEP -1
               1580 *      S(XL,Y+1)=S(XL,Y) : NEXT
               1590 *--------------------------------
0847- A4 02    1600 DOWN   LDY XL       COLUMN BEING MOVED DOWN
0849- A6 05    1610        LDX YB       BOTTOM CELL IN COLUMN
084B- CA       1620 .1     DEX
084C- 20 9D 08 1630        JSR MAKE.BASE
084F- B1 28    1640        LDA (BASE),Y
0851- 48       1650        PHA          SAVE CHAR IN CELL
0852- E8       1660        INX
0853- 20 9D 08 1670        JSR MAKE.BASE
0856- 68       1680        PLA
0857- 91 28    1690        STA (BASE),Y
0859- CA       1700        DEX
085A- E4 04    1710        CPX YT       AT TOP OF COLUMN YET?
085C- D0 ED    1720        BNE .1       NO
085E- 60       1730        RTS          YES
               1740 *--------------------------------
               1750 *      MOVE RIGHT SIDE UP
               1760 *      FOR Y=YT+1 TO YB
               1770 *      S(XR,Y-1)=S(XR,Y) : NEXT
               1780 *--------------------------------
```

```
085F- A4 03     1790 UP     LDY XR        COLUMN BEING MOVED UP
0861- A6 04     1800        LDX YT        TOP CELL IN COLUMN
0863- E8        1810 .1     INX
0864- 20 9D 08  1820        JSR MAKE.BASE
0867- B1 28     1830        LDA (BASE),Y
0869- 48        1840        PHA           SAVE CHAR IN CELL
086A- CA        1850        DEX           BACK UP
086B- 20 9D 08  1860        JSR MAKE.BASE
086E- 68        1870        PLA
086F- 91 28     1880        STA (BASE),Y
0871- E8        1890        INX
0872- E4 05     1900        CPX YB        AT BOTTOM OF COLUMN YET?
0874- D0 ED     1910        BNE .1        NO
0876- 60        1920        RTS           YES
                1930 *--------------------------------
                1940 *      MOVE TOP LINE LEFT
                1950 *      FOR X=XL+1 TO XR
                1960 *      S(X-1,YT)=S(X,YT) : NEXT
                1970 *--------------------------------
0877- A6 04     1980 LEFT   LDX YT        TOP LINE
0879- 20 9D 08  1990        JSR MAKE.BASE
087C- A4 02     2000        LDY XL
087E- C8        2010 .1     INY
087F- B1 28     2020        LDA (BASE),Y
0881- 88        2030        DEY
0882- 91 28     2040        STA (BASE),Y
0884- C8        2050        INY
0885- C4 03     2060        CPY XR        LAST COLUMN YET?
0887- D0 F5     2070        BNE .1        NO
0889- 60        2080        RTS
                2090 *--------------------------------
                2100 *      MOVE BOTTOM LINE RIGHT
                2110 *      FOR X=XR-1 TO XL STEP -1
                2120 *      S(X+1,YB)=S(X,YB) : NEXT
                2130 *--------------------------------
088A- A6 05     2140 RIGHT  LDX YB        BOTTOM LINE
088C- 20 9D 08  2150        JSR MAKE.BASE
088F- A4 03     2160        LDY XR
0891- 88        2170 .1     DEY
0892- B1 28     2180        LDA (BASE),Y
0894- C8        2190        INY
0895- 91 28     2200        STA (BASE),Y
0897- 88        2210        DEY
0898- C4 02     2220        CPY XL        FIRST COLUMN YET?
089A- D0 F5     2230        BNE .1        NO
089C- 60        2240        RTS
                2250
                2260 *--------------------------------
                2270 *      POINT BASE TO SCREEN LINE
                2280 *      (X) = LINE #
                2290 *--------------------------------
                2300 MAKE.BASE
                2310 *      TXA           ALTERNATE APPROACH
                2320 *      JMP $FBC1     USING BASCALC IN ROM
                2330
089D- BD A8 08  2340        LDA HI,X      FAST APROACH USING
08A0- 85 29     2350        STA BASE+1    TABLE LOOKUP
08A2- BD C0 08  2360        LDA LO,X
08A5- 85 28     2370        STA BASE
08A7- 60        2380        RTS
                2390 *--------------------------------
08A8- 04 04 05
08AB- 05 06 06
08AE- 07 07     2400 HI     .HS 0404050506060707
08B0- 04 04 05
08B3- 05 06 06
08B6- 07 07     2410        .HS 0404050506060707
08B8- 04 04 05
08BB- 05 06 06
08BE- 07 07     2420        .HS 0404050506060707
08C0- 00 80 00
08C3- 80 00 80
08C6- 00 80     2430 LO     .HS 0080008000800080
08C8- 28 A8 28
08CB- A8 28 A8
08CE- 28 A8     2440        .HS 28A828A828A828A8
08D0- 50 D0 50
08D3- D0 50 D0
08D6- 50 D0     2450        .HS 50D050D050D050D0
```

Breaking and Entering..............................Bill Morgan

We had a burglary here last night (today is May 26). Thieves got into the building somehow, and broke through the doors of several businesses, including your favorite software house and newsletter publisher.

They got three complete computer systems, including a two day old Apple //e that belonged to Judy Preston (she handles your orders), and an Apple /// system on loan from Apple Computer!

Just in case somebody tries to sell any of you some used Apple equipment, here's the list of what we lost, including serial numbers where known:

Apple][Plus Computer	1498251
STB 16K RAM Card	
Epson Printer Card	
Apple High-Speed Serial Card	
Wico Track Ball w/Interface	
Apple Disk Controller	
2 Apple Disk Drives	414611 & ?
Epson MX-80 Printer	
NEC Green Monitor	1315572
R-H Super Fan II	

Apple //e Computer	152413
Extended 80-Column Card	
Apple Disk Controller	
Apple Disk Drive	
Leedex B/W Monitor Video-100	804069

Apple /// Computer	14567
Apple /// Monitor	002073
Apple Silentype Printer	314027
Apple Disk ///	18157
TI Programmer Calculator (LCD Display)	

If you do see any of the above items, call your local police and/or S-C Software. Try to find out the name and address of the person selling the goods.

The thieves also took who-knows-how-many disks (at least three Flip-Files, two library cases, and many loose ones), containing the text and code from about the last three newsletters, all the disks from the Apple /// project (including the source code for the new assembler), several projects-in-progress, and whatever was handy. It will probably be months until we know what all is gone.

As I mentioned on the front page, the Spinwriter was damaged. They apparently got about half way down the hall carrying the printer, and then dropped it onto the concrete floor! We don't yet know what will be necessary to repair it.

Anyway, we're all alive and well, and we're going to carry on. See you next month!

Dear Assembly Line,

I hope you can answer a question I have concerning assembly language. I am just a beginner.

How can I convert hex numbers to decimal from within a running machine language program? That is, take bytes from locations A and A+1, convert the bytes to their decimal equivalent, and store the resulting ASCII bytes in other memory locations.

As a new member of A.P.P.L.E., I called their technical assistance line. They advised me to call Don Williams, who in turn directed me to Val Golding, editor of Call APPLE magazine. Val referred me to old issues of Call APPLE and Apple Orchard which I don't have available.

This appears to be a sticky question that nobody wants to deal with. There are a million ways to do this from BASIC, but I have yet to see one that will do it strictly within machine language.

(signed)
I. M. Perplexed
Anaheim, California

Dear Mr. Perplexed,

Your odyssey in search of a conversion program sounds as frustrating as it probably was! Let me assure you no one is trying to avoid dealing with these kinds of programs.

It is just that there are hundreds of variations. And many of them have been printed during the past five or six years in Micro, Nibble, Call APPLE, Apple Orchard, Kilobaud, Byte, and hundreds of Apple user group newsletters. All or most back issues of the major magazines are available through anthology volumes such as The Nibble Express, The Best of Micro, and Peeking at Call APPLE.

Also, almost all of the books written to teach 6502 assembly language programming include as examples subroutines to convert from binary to decimal and decimal to binary. We especially recommend Roger Wagner's "Assembly Lines--The Book" and Lance Leventhal's "6502 Subroutines". In fact, we even sell both of them at a slight discount.

I don't want to send you away looking for yet another source of information, so here is a routine I have used.

The subroutine assumes that you have placed the binary value into a variable called BINARY.VALUE, and places the converted result into DECIMAL.VALUE as a series of five ASCII characters. After conversion, the value in BINARY.VALUE will be zeroed.

This particular version of conversion produces leading zeroes for values below 10000. Variations I have used substitute leading blanks, or left justify with trailing blanks, or return a left justified value with a digit count.

The general method involves subtracting 10000 as many times as possible, and counting the times to get the first digit; subtracting 1000 from the remainder as mańy times as possible and counting the times to get the next digit; and so on. To simplify indexing, the constants 1, 10, 100, 1000, and 10000 are stored with the low-order bytes in one table, and the high-order bytes in another.

Line 1060 sets the Y-register to zero, for use as a pointer to the byte positions in DECIMAL.VALUE. If you were storing into a line buffer, you might enter th routine with the Y-register pointing to the starting place in the line and skip this initialization step.

Line 1070 sets X=4, which is one less than the number of digits you want to convert. X=4 gives five digits; if you are sure the value of the decimal number will be smaller, you can set X for fewer digits and enter past this point.

The loop from line 1090 through 1290 develops each digit in turn. Line 1090 starts a new digit by loading an ASCII zero. This value is pushed onto the stack during the subtraction that follows. Each time a subtraction is successful, it will be pulled off, incremented, and then pushed back on.

Lines 1120-1170 subtract the current divisor without storing the difference back into BINARY.VALUE. If the difference is positive, it is stored and the digit incremented. If the result is negative, then that was one subtraction too many, so the difference is discarded and the digit is retrieved from the stack at line 1250.

Line 1260 stores the converted digit into DECIMAL.VALUE, and line 1270 advances the digit pointer. Line 1280 advances the divisor pointer. If there are more divisors, line 1290 branches back to convert the next lower digit.

I have used a form of this subroutine inside the S-C Macro Assembler. That version includes options to also print the decimal value as it is being converted. Different entry conditions control whether the number will be printed, stored in a buffer, or both. I also allow control over the leading zeroes/blanks option and the number of digits.

This is only one method out of many, but it is fairly compact and easy to understand, without being too slow.

```
                 1010 *--------------------------------
                 1020 *      CONVERT VALUE TO DECIMAL CHARACTERS
                 1030 *      BY BOB SANDER-CEDERLOF
                 1040 *--------------------------------
                 1050 CONVERT
0800- A0 00      1060        LDY #0        POINT AT FIRST CHARACTER
0802- A2 04      1070        LDX #4        5 DIGITS
                 1080 *--------------------------------
0804- A9 B0      1090 .1     LDA #$B0      SET DIGIT TO ASCII ZERO
0806- 48         1100 .2     PHA           PUSH DIGIT ON STACK
0807- 38         1110        SEC           SUBTRACT CURRENT DIVISOR
0808- AD 37 08   1120        LDA BINARY.VALUE
080B- FD 2D 08   1130        SBC PLNTBL,X
080E- 48         1140        PHA           SAVE BYTE ON STACK
080F- AD 38 08   1150        LDA BINARY.VALUE+1
0812- FD 32 08   1160        SBC PLNTBH,X
0815- 90 0C      1170        BCC .3        LESS THAN DIVISOR
0817- 8D 38 08   1180        STA BINARY.VALUE+1
081A- 68         1190        PLA           GET LOW BYTE OFF STACK
081B- 8D 37 08   1200        STA BINARY.VALUE
081E- 68         1210        PLA           GET DIGIT FROM STACK
081F- 69 00      1220        ADC #0        INCREMENT DIGIT (CARRY WAS SET)
0821- D0 E3      1230        BNE .2        ...ALWAYS
0823- 68         1240 .3     PLA           DISCARD BYTE FROM STACK
0824- 68         1250        PLA           GET DIGIT FROM STACK
0825- 99 39 08   1260        STA DECIMAL.VALUE,Y
0828- C8         1270        INY           POINT TO NEXT DIGIT
0829- CA         1280 .4     DEX           POINT TO NEXT DIVISOR
082A- 10 D8      1290        BPL .1
082C- 60         1300        RTS           RETURN
                 1310 *--------------------------------
082D- 01         1320 PLNTBL .DA #1
082E- 0A         1330        .DA #10
082F- 64         1340        .DA #100
0830- E8         1350        .DA #1000
0831- 10         1360        .DA #10000
0832- 00         1370 PLNTBH .DA /1
0833- 00         1380        .DA /10
0834- 00         1390        .DA /100
0835- 03         1400        .DA /1000
0836- 27         1410        .DA /10000
                 1420 *--------------------------------
0837-            1430 BINARY.VALUE  .BS 2
0839-            1440 DECIMAL.VALUE .BS 5
                 1450 *--------------------------------
```

S-C Macro Assembler

S-C Software Corporation is pleased to introduce the S-C Macro Assembler, the latest version of our most popular product. The S-C Assembler II Version 4.0 already has the reputation of being the easiest editor/assembler to learn, to remember, and to use...now the S-C Macro Assembler provides a new level of power and performance for the beginner and professional programmer alike.

- 29 Commands, including a convenient EDIT command with 15 subcommands. COPY and REPLACE commands further simplify entry and modification of even the most complex programs.

20 Assembler Directives (Pseudo-Ops) provide all features necessary for professional software development, including conditional assembly and macro generation.

Operates in any Apple II or Apple II Plus with at least 32K RAM and one disk drive. Any additional memory or disk drives will be used as required. A Language Card version is also included.

A memory size of 48K allows source programs of over 24,000 bytes to be handled entirely within RAM. The Language Card version allows source programs of over 32,000 bytes. Much larger programs can be edited and assembled using the "INCLUDE" and "TARGET FILE" capabilities, up to the limit of on-line disk storage.

Programs can be edited, assembled, and tested entirely within the framework of the S-C Macro Assembler. The editor and assembler are co-resident, allowing rapid cycles of modification, re-assembly, and check-out. All DOS and Apple Monitor commands are active as well, providing a familiar interface to the standard Apple features.

Uses its own high-speed technique to store source files, but also can read or write standard TEXT files. You can EXEC in files from another assembler, use some other text editor to prepare files, keep a library of routines on disk to EXEC into any program, or use S-C Macro Assembler to prepare EXEC files for any purpose.

Price is only $80! Includes diskette with Macro Assembler and sample programs, a 100-page Reference Manual, and a Programmer Reference Card. (Registered Owners of S-C Assembler II Version 4.0 may purchase the upgrade package for only $27.50).

Commands

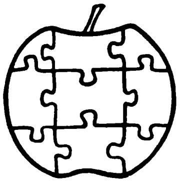
Source:	NEW, LOAD, SAVE, TEXT, HIDE, MERGE
Editing:	LIST, FIND, EDIT, DELETE, REPLACE, COPY, RENUMBER
List Control:	FAST, SLOW, PRT,"
Object:	ASM, MGO, VAL, SYMBOLS
Miscellaneous:	AUTO, MANUAL, INCREMENT, MEMORY, MNTR, RST, USR

All Aople Monitor Commands
All Apple DOS Commands

Assembler Directives

.OR	Origin
.TA	Target Address
.TF	Target File
.IN	Include File
.EN	End of Program
.EQ	Equate
.DA	1- or 2-byte Data
.HS	Hex String
.AS	ASCII String
.AT	ASCII Terminated
.BS	Block Storage
.TI	Title
.LIST	Listing Options
.PG	Page Eject
.DO	Conditional Assembly
.ELSE	
.FIN	
.MA	Macro Definition
.EM	End of Macro
.US	User Directive

Already well-known for excellent support, S-C Software Corporation pledges to continue development of new features, and to help owners gain the maximum benefit from the S-C Macro Assembler. In addition to telephone consultation for registered owners, a monthly newsletter is available by subscription (currently $15/year). The "Apple Assembly Line" covers items of interest to assembly language programmers at all levels, and has helped many to advance their programming skills.



"Makes assembly language programming on the Apple as easy as programming in BASIC."

S-C Software Corporation
2331 Gus Thomasson, Suite 125
P.O. Box 280300
Dallas, Texas 75228
(214) 324-2050
We take Master Card and Visa
Apple is a trademark of Apple Computer

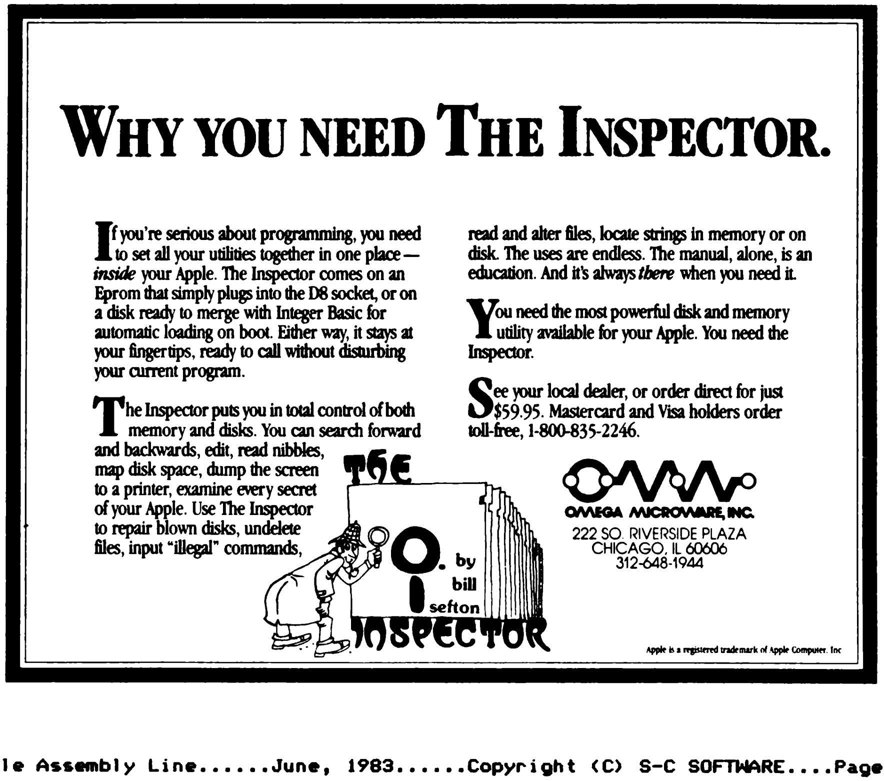
The Tiniest Motherboard......................Bob Sander-Cederlof

When I was at the Boston Applefest last week, Chad Pennebaker of Douglas Electronics showed me what he called the world's smallest Apple Mother Board. It is a bus board really, with ten Apple-Compatible 50-pin sockets labeled A, B, C, and 1 thru 7. The seven numbered slots accept almost any board made for Apples. Slot A is designed to hold a 6502 card, slot B a RAM card, and slot C a ROM card.

Some fantastic prices too: mother board, $95; CPU card, $90; 64K RAM card which maps with 48K from 0 to $BFFF, and 12K from $D000 to $FFFF, and another 4K from $D000 to $DFFF (sound familiar?), $190; 12K EPROM card (without EPROMS), $70.

Chad showed me a beautiful little cabinet it all fits in, Apple-beige metal with walnut sides. I didn't catch the price. There is also a power supply available, which looks just like the Apple unit. A card which provides keyboard and screen functions is on the way.

Here's how to reach them: Douglas Electronics, Inc., 718 Marina Blvd., San Leandro, CA 94577. Or call at (415) 483-8770.



Replacing INIT Can Be Dangerous.....................Bill Morgan

I have recently spent several days beating my head against an impossible bug, and I'll bet that some of you have run into, or will hit, the same thing. Here's my tale....

One of the most common enhancements to Apple's DOS is the addition of new commands. You do this by replacing one of the existing commands, usually INIT, VERIFY, or MAXFILES, with new code and a new name. I got into trouble replacing INIT, and had a great time figuring out why!

The SHOW Command ...

I was working with the SHOW command, as published in the July, 1982 Apple Assembly Line. Typing SHOW <filename> displays any sequential text file on the screen; it's a really handy command to have.

SHOW is installed in place of INIT. Making the change involves placing the new code over the File Manager's INIT handler at $AE8E and at $A54F, changing the command name at $A884, and altering the table of permissible and required keywords at $A909. These are the usual steps to replace a command. See the July '82 issue for more details about SHOW.

... and File Manager Calls

The S-C Word Processor uses one of the popular high-speed methods to handle text files. It calls the File Manager to OPEN the file, then uses its own code to LOAD or SAVE text, by directly calling RWTS. So far, so good, this is basically normal. However, after I installed the SHOW command in DOS, I couldn't SAVE new files from the Word Processor! SAVEing to an existing file worked just fine, but trying to create a new file returned a FILE NOT FOUND error. What???

A little bit of digging convinced me that this was impossible. What does changing INIT have to do with OPENing a text file? There is that table which tells what a command can or cannot do, but SHOW is nowhere near OPEN, and besides, the table is used by the command handler, not by the File Manager. You tell the File Manager that it can create a file, if necessary, by setting the X register to zero before doing the JSR $3D6. The Word Processor does that just right.

A lot more digging brought the solution to light. When the File Manager (FM) OPENs a file, it does refer to that table at $A909 to see if it can create a new file. It loads the X register with a command index kept at $AA5F, then checks the corresponding table entry. But what does that have to do with INIT? Read on...

When you enter FM through $3D6 it jumps to a special entry at $AAFD. The first thing it does there is check the X register to see if it will be allowed to create a new file. If X is nonzero (no new file), FM stores a 2 in the command index ($AA5F). That is the index for the LOAD command, which cannot create a file. If X is zero (new file allowed), FM stores that zero in the command index. And zero is the index to the INIT command, which does create a new file (usually HELLO).

So there it is. If we replace INIT with a command that is not allowed to create a new file, we will mess up programs that call the File Manager directly to OPEN new files. Ouch!!

Fixing the Problem

There are several possible ways to avoid such trouble:

1) Don't mess with INIT. If you leave it alone, it won't bite you. But, INIT is so useless in a running program and offers so much space for new code. It's sure hard to resist.

2) Only replace INIT with other commands that are allowed to create new files. That's OK, but limiting.

3) Add these additional patches to whatever new command you're playing with:

   A. Put JMP FM.ENTRY.PATCH at $AAFD.

   B. And put FM.ENTRY.PATCH wherever is convenient (in SHOW I put it at $AEA5, just after the PAUSE.CHECK code):

```
FM.ENTRY.PATCH
       CPX #0        Create new file?
       BEQ .1        0 means yes
       LDX #2        No, use LOAD index
       BNE .2        ... Always
.1     LDX #4        Yes, use SAVE index
.2     JMP $AB03     Return to File Manager
```

Conclusion

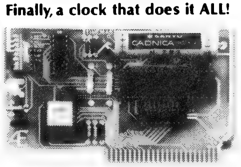
This kind of problem is a great example of why you need to be very careful about patching an operating system: it's hard to tell what kind of "impossible" interactions will turn up. On the other hand, how else can we learn about what's really going on inside our Apples? And what else can replace the "AHA!!! Ahhh..." sensation you get when you unravel a really cute bug? Keep on patchin'.

Apple Peripherals Are All We Make

That's Why We're So Good At It!

NEW

The TIMEMASTER
Finally, a clock that does it ALL!



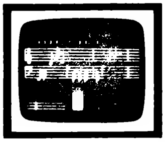
- Designed in 1983 using I.C. technologies that simply did not exist when most other Apple clocks were designed.
- Just plug it in and your programs can read the year, month, date, day, and time — down to 1 millisecond!
- Powerful 2K ROM driver — No clock could be easier to use.
- Full emulation of most other clocks, including Mountain Hardware's Appleclock (but you'll like the TIMEMASTER mode better).
- Compatible with all of Apple's languages, CP/M and PASCAL software on disk.
- Eight software controlled interrupts so you can execute two programs at the same time. (Many examples are included)
- On board timer lets you time any interval up to 48 days long down to the nearest millisecond.

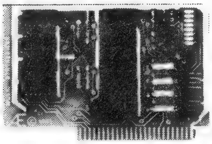
The TIMEMASTER includes a disk with some really fantastic time oriented programs (over 25) plus a DOS dater so it will automatically add the date when disk files are created or modified. This disk is over a $200.00 value alone — we give the software others sell. All software packages for business, data base management and communications are made to read the TIMEMASTER.

If you want the most powerful and the easiest to use clock for your Apple, you want a TIMEMASTER.

PRICE $129.00

Super Music Synthesizer



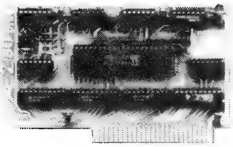


- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, the disk is filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Will play songs written for ALF synthesizer (ALF software will not take advantage of all the features of this board. Their software sounds the same in our synthesizer.)
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE $159.00

NEW

Z-80 PLUS!



- TOTALLY compatible with ALL CP/M software.
- Executes the full Z-80 and 8080 instruction set.
- Fully compatible with microsoft disks (no pre-boot required).
- An on-card PROM eliminates many I.C.'s for a cooler, less power consuming board. (We use the Z-80A at a fast 3.58 MHZ)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts.
- All new 1983 design incorporates the latest in I.C. technologies.
- Complete documentation included. (User must furnish software)

The Z-80 PLUS turns your Apple into a CP/M based computer. This means you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

COMING SOON: The Z-80 Plus for the Apple III

PRICE $139.00

Analog to Digital Converter

- 8 Channels
- 8 Bit Resolution
- On Board Memory
- Fast Conversion (.078 ms per channel)
- Eliminates the Need to Wait for A/D Conversion (just PEEK at data)
- A/D Process Totally Transparent to Apple (looks like memory)

The analog to digital conversion takes place on a continuous, channel sequencing basis. Data is automatically transferred to on board memory at the end of each conversion. No A/D converter could be easier to use.

Our A/D board comes standard with 0, 10V full scale inputs. These inputs can be changed by the user to 0, -10V, or -5V, +5V or other ranges as needed.

Information on temperature sensors is given in manual.

The user connector has +12 and -12 volts on it so you can power your sensors.

Accuracy 0.3% Input Resistance 20K Ohms Typ

A few applications may include monitoring and control of ● flow ● temperature ● humidity ● wind speed ● wind direction ● light intensity ● pressure ● RPM ● storage oscilloscope ● soil moisture and many more.

We also manufacture a 16 channel digital input/output board for control applications.

PRICE $129.00

NEW

IIe Only: 80 Column, 64K RAM Card

- Expand your Apple IIe to 128K memory.
- Provides an 80 column text display.
- TOTALLY compatible with ALL Apple software and languages, there are NO exceptions.
- Automatically expands VisiCalc to 95K storage. In 80 columns!
- COMPLETE documentation included. (We don't make you refer to the Apple manual as others do.)
- Uses the same commands as the Apple 80 column board.
- Incorporates the latest high speed, low power I.C. technologies.
- Plugs into the Apple IIe expansion slot.
- Simply the best expansion card for your Apple IIe at any price, offering you phenomenal performance at a very nominal price.

PRICE $149.00

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in APPLE IIe, II and II+. (Except 80 column card.)

Applied Engineering's products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle two year warranty.

All Orders Shipped Same Day
Texas Residents Add 5% Sales Tax
Add $10.00 If Outside U.S.A.
Dealer Inquiries Welcome

Send Check or Money Order to:
APPLIED ENGINEERING
P.O. Box 470301
Dallas, TX 75247

Call (214) 492-2027
7am to 11pm 7 days a week
MasterCard, Visa & C.O.D. Welcome

Reformatting a Lot of Text.................Bob Sander-Cederlof

At the Boston AppleFest I picked up a copy of David Durkee's SoftGraph program. You probably read the series of four articles in Softalk (Jan thru Apr, 1983) in which he developed this fine little system for editing data and creating pie, bar, and line charts. If you didn't, let me recommend them.

(If you don't get Softalk, why not? It's free! The best magazine there is for Apple owners! Send your serial number to Al Tommervik at Softalk, Box 60, North Hollywood, CA 91603 today!)

Anyway, back to SoftGraph. On the disk is a 107 sector binary file loaded with ASCII characters. A small program on the disk will display or print the text from this file. Never satisfied with things as they come, I wanted to load the file into my word processor. The problem itself may be irrelevant to you, but the steps to solving it can be quite instructive. Follow along now....

My word processor will read binary or text files, but it expects the data to be ASCII with the high bits all set to 1. Naturally, Durkee's file had all high bits set to 0. "That's OK, I'll just run the handy little code from AAL Dec 82, 'Add Bit-Control to Apple Monitor' "

So I did, but setting all the high bits wasn't quite enough. A second feature of the file came to light: no carriage returns anywhere. Each line was padded with trailing blanks to fill exactly 40 bytes. Even the blank lines contained 40 blanks. I fidgeted in my chair, and pulled a little hair.

After several false starts I finally resorted to a trick I learned back in the dark ages before structured programming, with its scientific rules and esoteric vocabulary, was invented. I constructed a flow chart! I do this every once and a while, as a thinking tool. But you probably won't find any in my documentation, because they are just a tool. I usually modify my thinking as I code, which obseletes the chart. Most charts are done on odd bits of scrap paper, and don't last overnight.

Here was my plan. First, BLOAD the file somewhere in memory and find its end by looking at $AA60 and $AA61. This pair of bytes contains the length of the last file loaded. Second, run a conversion program to change the data IN PLACE. Third, BSAVE the modified data on a new file. I decided to save time by manually typing the BLOAD and BSAVE commands, rather than writing code to do these steps inside my conversion program.

It so happened that the file would fit between $2000 and $8977. With the S-C Assembler in the language card at $D000, this space was available. The source code for my conversion program fit above $9000 running up to $95FF. The object code started at $800, and didn't make it out of that page.

Without drawing the flow chart for you, here is the general idea of the conversion process:

1. Set up two pointers, one for retrieving characters out of the text and the other for storing converted characters into the text.

2. Get 40 characters from the text into a little buffer. If at the end (marked by a 00 byte), quit.

3. Scan backwards in the little buffer to the first non-blank character.

4. If the whole line is blank, store a carriage return into the text.

5. Otherwise, copy the little buffer back into the text, with the high bit set on each byte. Then add one blank, because we need to maintain a blank between words. (They may not print on separate lines after re-formatting with my word processor.)

6. Back to 2.

(I'm sorry, but except for the lines and boxes, that does look a lot like a flow chart.)

One little wrinkle I thought about but didn't implement until later had to do with double spacing between paragraphs. I handled it by checking the length of the previous line after stuffing the carriage return for a blank line. If the previous line was non-blank, I sent an extra carriage return back into the text.

My routine counts on the assumed condition that the resulting text will be shorter than the original text. I knew the file began with several lines of 40 blanks, each being replaced with a single carriage return, so I felt confident that all would work. If I was wrong, I would be storing modified data on top of yet-unprocessed data, with wild results. Don't worry, it worked out OK.

Lines 1050-1090 define some variables. The data will begin at $2000, because I put it there with a "BLOAD DOCFILE,A$2000" command. GET.PNTR and PUT.PNTR will start out pointing at $2000. Each time I pull 40 characters out of the data I will add 40 to GET.PNTR. Each time I put one character back into the data I will add one to PUT.PNTR. LAST.LINE.SZ keeps track of previous line length so I can get double spacing between paragraphs.

BUFFER is for the forty characters pulled out of the data each cycle through the program. I frequently use $200 for buffers like this, because it is a nice handy area. And most Apple software uses $200 for a buffer. But...since the monitor does use $200, it can be difficult to see what is put there by my program. Hence, this time I put the buffer at $280 instead.

Lines 1110-1390 implement the logic flow described above. Previous line length starts out zero when there were no previous lines (1120-1130). The two pointers get initialized at 1140-1190.

Calling GET.40.CHARS pulls the next 40 bytes out of the data into my buffer at $280. If a 00 byte is hit, the subroutine returns with carry set; if not, carry is clear. Line 1210 acts on the carry info to end the program if we are done.

TRUNCATE.BLANKS starts at the end of the buffer looking for non-blanks. If the whole buffer is blank, the subroutine sets carry. If not, it clears carry. Line 1230 acts on carry. Non-blank lines are copied back into the data by PUT.CHARS, and then PUT.CHAR is used to add one trailing blank. Blank lines cause a return ($8D) to be put into the data, and if the previous line was non-blank a second return is added.

PUT.CHAR (lines 1410-1470) stores the byte in the A-register into the data where PUT.PNTR points. Then PUT.PNTR is incremented. PUT.CHARS (lines 1810-1880) calls PUT.CHAR once for each character in the buffer, omitting all the trailing blanks.

With trepidation I typed $800G to execute it, after being sure I had saved the source code and then opened both disk drive doors. To my surprise, the program ran without failure the very first time! Not to say it was perfect.

After execution I looked at PUT.PNTR to see where the new data ended. It was $70C7, if I remember correctly. Then I BSAVE'd with "BSAVE DOCFILE 2,A$2000,L$50C7", and loaded my word processor.

In the word processor I loaded the new DOCFILE 2, and it was all there. Somehow two small sections were missing all the carriage returns, but all the rest was perfect. I still don't know what caused those two sections (total of about ten lines) to fail, but it isn't all that important. I used the word processor to fix them, and the job was done.

```
                  1000 *SAVE S.CONVERT DURKEE
                  1010 *--------------------------------
                  1020 *      CONVERT DURKEE'S DOCFILE
                  1030 *      TO S-C WORD ASCII FORMAT
                  1040 *--------------------------------
2000-             1050 DATA         .EQ $2000
0000-             1060 GET.PNTR     .EQ $00,01
0002-             1070 PUT.PNTR     .EQ $02,03
0004-             1080 LAST.LINE.SZ .EQ $04
0280-             1090 BUFFER       .EQ $280
                  1100 *--------------------------------
```

```
               1110 CONVERT
0800- A9 00    1120        LDA #0
0802- 85 04    1130        STA LAST.LINE.SZ
0804- A9 00    1140        LDA #DATA
0806- 85 00    1150        STA GET.PNTR
0808- 85 02    1160        STA PUT.PNTR
080A- A9 20    1170        LDA /DATA
080C- 85 01    1180        STA GET.PNTR+1
080E- 85 03    1190        STA PUT.PNTR+1
0810- 20 49 08 1200 .1     JSR GET.40.CHARS
0813- B0 23    1210        BCS .3
0815- 20 69 08 1220        JSR TRUNCATE.BLANKS
0818- B0 0A    1230        BCS .2        EMPTY LINE
081A- 20 79 08 1240        JSR PUT.CHARS
081D- A9 A0    1250        LDA #$A0      BLANK
081F- 20 3E 08 1260        JSR PUT.CHAR
0822- D0 EC    1270        BNE .1        ...ALWAYS
0824- A9 8D    1280 .2     LDA #$8D      <RETURN>
0826- 20 3E 08 1290        JSR PUT.CHAR
0829- A5 04    1300        LDA LAST.LINE.SZ
082B- F0 E3    1310        BEQ .1
082D- A9 00    1320        LDA #0
082F- 85 04    1330        STA LAST.LINE.SZ
0831- A9 8D    1340        LDA #$8D
0833- 20 3E 08 1350        JSR PUT.CHAR
0836- D0 D8    1360        BNE .1        ...ALWAYS
0838- A9 00    1370 .3     LDA #0        <EOL>
083A- 20 3E 08 1380        JSR PUT.CHAR
083D- 60       1390        RTS
               1400 *--------------------------------
               1410 PUT.CHAR
083E- A0 00    1420        LDY #0
0840- 91 02    1430        STA (PUT.PNTR),Y
0842- E6 02    1440        INC PUT.PNTR
0844- D0 02    1450        BNE .1
0846- E6 03    1460        INC PUT.PNTR+1
0848- 60       1470 .1     RTS
               1480 *--------------------------------
               1490 GET.40.CHARS
0849- A0 00    1500        LDY #0
084B- B1 00    1510 .1     LDA (GET.PNTR),Y
084D- C9 00    1520        CMP #0
084F- F0 16    1530        BEQ .3        END OF DATA
0851- 09 80    1540        ORA #$80
0853- 99 80 02 1550        STA BUFFER,Y
0856- C8       1560        INY
0857- C0 28    1570        CPY #40
0859- 90 F0    1580        BCC .1
085B- A9 27    1590        LDA #39       CARRY SET
085D- 65 00    1600        ADC GET.PNTR
085F- 85 00    1610        STA GET.PNTR
0861- 90 02    1620        BCC .2
0863- E6 01    1630        INC GET.PNTR+1
0865- 18       1640 .2     CLC
0866- 60       1650        RTS
0867- 38       1660 .3     SEC           END OF DATA
0868- 60       1670        RTS
               1680 *--------------------------------
               1690 TRUNCATE.BLANKS
0869- B9 7F 02 1700 .1     LDA BUFFER-1,Y
086C- C9 A0    1710        CMP #$A0      BLANK?
086E- D0 05    1720        BNE .2        NO
0870- 88       1730        DEY           YES
0871- D0 F6    1740        BNE .1
0873- 38       1750        SEC           EMPTY LINE
0874- 60       1760        RTS
0875- 84 04    1770 .2     STY LAST.LINE.SZ
0877- 18       1780        CLC
0878- 60       1790        RTS
               1800 *--------------------------------
               1810 PUT.CHARS
0879- A2 00    1820        LDX #0
087B- BD 80 02 1830 .1     LDA BUFFER,X
087E- 20 3E 08 1840        JSR PUT.CHAR
0881- E8       1850        INX
0882- E4 04    1860        CPX LAST.LINE.SZ
0884- 90 F5    1870        BCC .1
0886- 60       1880        RTS
```

Track Balls...Bill Morgan

If you have ever played Centipede or Missile Command in the arcade, then you know that a track ball is about the best control device made. A joystick usually can move across the whole screen a little faster, but a track ball gives much finer, smoother control. A "mouse" is said to be even better, but the only mouse I have seen advertised for the Apple][sells for about $300-400. Besides, I never have 2-3 square feet of free space on my desk, for the mouse to run around on.

Several track balls for the Apple have appeared recently, all in the $60-80 price range. I have tried out two of them so far, from TG Products and from Wico Corp. Here's what I think:

The TG track ball is an Apple-colored box about 5 x 6 x 2 1/2 inches, with a red ball that is a little over two inches in diameter. There are two pushbuttons on the left edge of the box. It plugs into the game port, just like a joystick. It contains two potentiometers and can be used with existing paddle-reading software, also just like a joystick. This ball feels stiff and jerky, and requires a constant downward pressure on the ball to keep the pots properly tracking. The range of values is 0-255, with no wraparound.

$64.95 TG Products, 1104 Summit Ave., #110, Plano, TX, 75074

The Wico track ball is a cream-colored ball in a black and red box, just about the same size as the TG. There are two buttons in the upper left corner of the top side, convenient to the left thumb. The larger button is about .8" in diameter, the other is about .3" across. This unit uses its own interface card (which is supplied), so it leaves the game port free, but requires a motherboard slot.

Wico's ball is based on the same design as the arcade controls: the track ball rolls on ball bearings, and is read by optically counting the revolutions of the rollers supporting the ball. This design gives a much better, smoother feel to the ball's motion, and gives the programmer more flexible ways to read and control the ball. However, it also means that no existing programs can use the Wico ball.

$79.95 Wico Corp., 6400 Gross Point Rd., Niles, IL, 60648

In summary, the TG track ball fits right into the "standard" Apple environment. It plugs into the game port and works with all software that reads paddles 0 and 1, but it feels awkward to use. I consider it a poor substitute for a joystick or paddles, where they are appropriate ... and a poor substitute for a real trackball, if that's what you need.

On the other hand, the Wico track ball is much more responsive and comfortable to use, but it requires an interface slot and special programming. I think it's well worth the effort, and I intend to try using it in as many different applications as I can think of.

Wico's trackball comes with a booklet containing a couple of pages about programming with their interface. The following is a summary of that information, plus whatever I've been able to figure out. The program given here reads the ball and displays the X and Y values on the screen in hex notation. It also checks the keyboard for the keys "1" through "4", and sets the ball's speed to match.

The interface card contains no ROM. It does have eight registers which you read and/or write to control the trackball. Here is a table of the registers' addresses and functions ("N" is slot number + 8, i.e., $9-$F):

Address	Read	Write
$C0N0	X Position	X Position
$C0N1	Y Position	Y Position
$C0N2	Bounded	Bounded
$C0N3	Wraparound	Wraparound
$C0N4	-	Speed 1
$C0N5	-	Speed 2
$C0N6	Buttons	Speed 3
$C0N7	-	Speed 4

The first two registers, $C0N0 and $C0N1, contain the X and Y readings from the trackball. You can write to these locations to set starting values, or to force particular values at any time. Lines 1850-2040 of the program show how to read the registers, limit their values, and keep track of current value, last value and change since last reading.

Another approach is to read only the change in value from the trackball, and keep track of the values separately. To do that, first turn on the wraparound feature, as described below. Then set $C0N0 and $C0N1 to 0. That takes care of initialization. Now, whenever you want to read the changes in the ball's position, just call this routine:

```
LDA XREG
STA DX
LDA YREG
STA DY
LDA #0
STA XREG
STA YREG
RTS
```

Since wraparound is permitted, DX and DY will be positive when the ball was moved down or right, and negative when it was moved up or left. Reset the registers to 0 after reading them, and next time you call this routine they will again contain only the change in value.

$C0N2 (BOUNDRY) and $C0N3 (WRAPS) control whether the readouts will stop at 0 and 255, or wrap around. Reading or writing to either address will set the corresponding condition.

You can read the state of the pushbuttons from $C0N6; each button turns on one bit. Bit 7 (sign bit) is the large button and bit 6 (overflow bit) is the small one. These are very easy to test from assembly language; just BIT $C0N6 and use BMI & BPL for the large button, or BVC & BVS for the small one. Lines 2060-2140 of the program show a good way to translate these bits into bytes, so Applesoft can easily test them.

The speed or scale of the readout can be controlled by writing to two of the locations from $C0N4-$C0N7. The values written do not matter, you're throwing soft switches. These addresses select a divider to apply to the X and Y readings. Here's a table of addresses and effects:

Addresses	Speed	Divide by
$C0N6 & $C0N4	Fastest	1
$C0N6 & $C0N5	Med Fast	2
$C0N7 & $C0N4	Med Slow	4
$C0N7 & $C0N5	Slowest	8

At the fastest setting, a quarter-turn of the ball produces about a sixteen-point difference in the X or Y reading. At the slowest setting, the same motion changes the readout by two points. Lines 2160-2260 are just a quick way to read the keyboard and produce a value of 0-3. Lines 2280-2390 are how I translate that number into writing the correct pair of addresses.

The track ball has also proved to be an excellent cursor control for graphics work, and a lot of fun to use for controlling menu selection. I'm looking forward to trying it out as a cursor control with the S-C Word Processor.

We don't plan to stock the Track Balls, but if you want one, we can get the Wico unit for you for $75 plus shipping.

```
                    1000 *--------------------------------
                    1010 *  READ AND WRITE WICO TRACKBALL INTERFACE
                    1020 *--------------------------------
0024-               1030 CH       .EQ $24
C000-               1040 KEYBOARD .EQ $C000
C010-               1050 STROBE   .EQ $C010
FC24-               1060 VTABZ    .EQ $FC24
FC58-               1070 HOME     .EQ $FC58
FDED-               1080 COUT     .EQ $FDED
FDDA-               1090 PRBYTE   .EQ $FDDA
                    1100 *--------------------------------
                    1110 *    WICO INTERFACE REGISTERS
                    1120
0004-               1130 SLOT     .EQ 4     INTERFACE LOCATION
                    1140
C0C0-               1150 BASE     .EQ SLOT*$10+$C080
C0C0-               1160 REGS     .EQ BASE+0
C0C0-               1170 XREG     .EQ BASE+0
C0C1-               1180 YREG     .EQ BASE+1
C0C2-               1190 BOUNDRY  .EQ BASE+2
C0C3-               1200 WRAP     .EQ BASE+3
C0C4-               1210 SPEED1   .EQ BASE+4
C0C5-               1220 SPEED2   .EQ BASE+5
C0C6-               1230 SPEED3   .EQ BASE+6
C0C7-               1240 SPEED4   .EQ BASE+7
C0C6-               1250 BUTTONS  .EQ BASE+6
                    1260 *--------------------------------
```

```
0800- 20 58 FC 1270 SETUP   JSR HOME
0803- A9 00    1280         LDA #0
0805- 8D 4F 08 1290         STA SPEED          START AT TOP SPEED
0808- 8D C2 C0 1300         STA BOUNDRY        NO WRAPAROUND
080B- A0 01    1310         LDY #1             DO THIS TWICE
080D- B9 45 08 1320 .1      LDA HIGH.LIMITS,Y
0810- F9 43 08 1330         SBC LOW.LIMITS,Y   SET INITIAL
0813- 4A       1340         LSR                VALUE TO
0814- 79 43 08 1350         ADC LOW.LIMITS,Y   CENTER OF
0817- 99 47 08 1360         STA LOCATIONS,Y    LIMITS
081A- 99 C0 C0 1370         STA REGS,Y
081D- 99 49 08 1380         STA LAST.VALUES,Y
0820- 88       1390         DEY
0821- 10 EA    1400         BPL .1             DONE?
               1410
0823- A9 0A    1420 LOOP    LDA #10            CENTER DISPLAY
0825- 20 24 FC 1430         JSR VTABZ
0828- A9 10    1440         LDA #16            ON SCREEN
082A- 85 24    1450         STA CH
082C- 20 50 08 1460         JSR READ.BALL      GO READ BALL
082F- AD 47 08 1470         LDA X
0832- 20 DA FD 1480         JSR PRBYTE         SHOW X READING
0835- A9 A0    1490         LDA #$A0
0837- 20 ED FD 1500         JSR COUT
083A- AD 48 08 1510         LDA Y              AND Y READING
083D- 20 DA FD 1520         JSR PRBYTE
0840- 4C 23 08 1530         JMP LOOP           DO IT AGAIN
               1540 *--------------------------------
               1550 *   VARIABLES
               1560 *--------------------------------
               1570 LOW.LIMITS
0843- 00       1580 X.LOW   .DA #0        MINIMUM VALUES
0844- 00       1590 Y.LOW   .DA #0
               1600
               1610 HIGH.LIMITS
0845- FF       1620 X.HIGH  .DA #$FF      MAXIMUM VALUES
0846- FF       1630 Y.HIGH  .DA #$FF
               1640
               1650 LOCATIONS
0847- 00       1660 X       .DA #0        POINTS TO PLOT
0848- 00       1670 Y       .DA #0
               1680
               1690 LAST.VALUES
0849- 00       1700 LAST.X  .DA #0        FROM LAST CALL
084A- 00       1710 LAST.Y  .DA #0
               1720
               1730 DELTAS
084B- 00       1740 DX      .DA #0        CHANGE IN VALUES
084C- 00       1750 DY      .DA #0
               1760
084D- 00       1770 S1      .DA #0        LARGE BUTTON
084E- 00       1780 S2      .DA #0        SMALL BUTTON
               1790
084F- 00       1800 SPEED   .DA #0        0=FASTEST, 3=SLOWEST
               1810
               1820 *--------------------------------
               1830 READ.BALL
               1840
               1850 SET.X.AND.Y
0850- A0 01    1860         LDY #1             DO THIS TWICE
0852- B9 C0 C0 1870 .1      LDA REGS,Y         READ BALL
0855- D9 43 08 1880         CMP LOW.LIMITS,Y   TOO LOW?
0858- B0 06    1890         BCS .2             NO, GO ON
085A- B9 43 08 1900         LDA LOW.LIMITS,Y   YES, FORCE LOW LIMIT
085D- 99 C0 C0 1910         STA REGS,Y
0860- D9 45 08 1920 .2      CMP HIGH.LIMITS,Y  TOO HIGH?
0863- 90 06    1930         BCC .3             NO, GO ON
0865- B9 45 08 1940         LDA HIGH.LIMITS,Y  YES, FORCE HIGH LIMIT
0868- 99 C0 C0 1950         STA REGS,Y
086B- 99 47 08 1960 .3      STA LOCATIONS,Y    USE THIS POINT
086E- 48       1970         PHA                SAVE IT
086F- 38       1980         SEC
0870- F9 49 08 1990         SBC LAST.VALUES,Y  CALCULATE CHANGE
0873- 99 4B 08 2000         STA DELTAS,Y
0876- 68       2010         PLA                RESTORE POINT USED
0877- 99 49 08 2020         STA LAST.VALUES,Y  AND SAVE IT FOR NEXT READING
087A- 88       2030         DEY
087B- 10 D5    2040         BPL .1             DONE?
```

```
                 2050
                 2060 SET.SWITCHES
087D- A9 00      2070        LDA #0
087F- 8D 4D 08   2080        STA S1        ZERO
0882- 8D 4E 08   2090        STA S2        READOUTS
0885- AD C6 C0   2100        LDA BUTTONS   READ PUSHBUTTONS
0888- 0A         2110        ASL           BIT 7 TO CARRY
0889- 2E 4D 08   2120        ROL S1          TO S1.
088C- 0A         2130        ASL           BIT 6 TO CARRY
088D- 2E 4E 08   2140        ROL S2          TO S2.
                 2150
                 2160 CHECK.KEYBOARD
0890- AD 00 C0   2170        LDA KEYBOARD    KEYPRESS?
0893- 10 24      2180        BPL EXIT        NO, GO ON
0895- 8D 10 C0   2190        STA STROBE      YES
0898- C9 B5      2200        CMP #$B5        >4?
089A- B0 1D      2210        BCS EXIT        YES, GO ON
089C- C9 B0      2220        CMP #$B0        <1?
089E- 90 19      2230        BCC EXIT        YES, GO ON
08A0- 29 0F      2240        AND #$0F        LOSE HIGH NYBBLE
08A2- E9 01      2250        SBC #1          MAKE 0-3
08A4- 8D 4F 08   2260        STA SPEED       AND SAVE IT
                 2270
                 2280 SET.SPEED
08A7- AD 4F 08   2290        LDA SPEED     GET SPEED
08AA- 48         2300        PHA
08AB- 29 02      2310        AND #2        USE BIT 1
08AD- 4A         2320        LSR           NOW 0 OR 1
08AE- A8         2330        TAY           INDEX
08AF- 99 C6 C0   2340        STA SPEED3,Y  HIT SPEED3 OR SPEED4
08B2- 68         2350        PLA           GET SPEED AGAIN
08B3- 29 01      2360        AND #1        USE BIT 0
08B5- A8         2370        TAY
08B6- 99 C4 C0   2380        STA SPEED1,Y  HIT SPEED1 OR SPEED2
08B9- 60         2390 EXIT   RTS
                 2400 *--------------------------------
```

S-C Macro Cross Assemblers

The high cost of dedicated microprocessor development systems has forced many technical people to look for alternate methods to develop programs for the various popular microprocessors. Combining the versatile Apple II with the S-C Macro Assembler provides a cost effective and powerful development system. Hobbyists and engineers alike will find the friendly combination the easiest and best way to tend their skills to other microprocessors.

The S-C Macro Cross Assemblers are all identical in operation to the S-C Macro Assembler; only the language assembled is different. They are sold as upgrade packages to the S-C Macro Assembler. The S-C Macro Assembler, complete with 100-page reference manual, costs $80; once you have it, you may add as many Cross Assemblers as you wish at a nominal price. The following S-C Macro Cross Assembler versions are now available, or soon will be:

Motorola:	6800/6801/6802	now	$32.50
	6805	now	$32.50
	6809	now	$32.50
	68000	now	$50.00
Intel:	8048	now	$32.50
	8051	now	$32.50
	8085	soon	$32.50
Zilog:	Z-80	now	$32.50
RCA:	1802/1805	now	$32.50
Rockwell:	65C02	now	$20.00
DEC:	PDP-11/LSI-11	now	$50.00

The S-C Macro Assembler family is well known for its ease-of-use and powerful features. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependablility, and user-friendliness. There are 20 assembler directives to provide powerful macros, conditional assembly, and flexible data generation. INCLUDE and TARGET FILE capabilities allow source programs to be as large as your disk space. The integrated, co-resident source program editor provides global search and replace, move, and edit. The EDIT command has 15 sub-commands combined with global selection.

Each S-C Assembler diskette contains two complete ready-to-run assemblers: one is for execution in the mother-board RAM; the other executes in a 16K RAM Card. The HELLO program offers menu selection to load the version you desire. The disks may be copied using any standard Apple disk copy program, and copies of the assembler may be BSAVEd on your working disks.

-C Software Corporation has frequently been commended for outstanding upport: competent telephone help, a monthly (by subscription) newsletter, continuing enhancements, and excellent upgrade policies.

S-C Software Corporation (214) 324-2050
P.O. Box 280300, Dallas, Texas, 75228

Ampersand Monitor Caller..................Bob Sander-Cederlof

A recent issue of the Maple Orchard, magazine published by the Loyal Ontario Group Interested in Computers (LOGIC, P. O. Box 696 station B, Willowdale, Ontario, Canada M2K 2P9) was entirely devoted to assembly language. Bob Stitt, also one of our readers, authored an article called "A New Utility for Applesoft".

Bob's new utility provided a way to execute monitor commands from inside a running Applesoft program. He implemented the S.H.Lam method in machine language as an ampersand routine.

A long long time ago someone named S. H. Lam showed the world a neat way to execute monitor commands from Applesoft. His method was published by Call APPLE back about 1978, I think. Lam POKEd the characters of a string containing a monitor command into the monitor's input buffer at $200. He included " N D9C6G" at the end of each command string, to return control to Applesoft. Then POKE 72,0:CALL-144 executes the command.

```
     100 C$="300:A9 3A 20 C0 DE 60 N D9C6G"
     110 FOR I=1 TO LEN(C$)
     120 POKE 511+I, ASC ( MID$ (C$,I,1)) + 128
     130 NEXT
     140 POKE 72,0 : CALL-144
```

Bob Stitt's utility replaces lines 110-140 above with a simple ampersand statement:

```
     110 & C$
```

After reading the article, I decided to try writing my own. I came up with a different technique; it is probably no better, but it is bigger. Mine works like a similar routine coded by Steve Wozniak inside the mini-assembler in the Integer BASIC ROMs. The only advantage I find is that there is no need to append " N D9C6G" to each command string. As a result, you can execute "3D0G" (if you like) and stop execution of your Applesoft program.

Lines 1220-1280 set up the ampersand vector. You can BLOAD the program and CALL 768 to run these lines, or simply BRUN it.

Line 1310 clears the monitor MODE, so that it realizes it is at the beginning of a command.

Lines 1320 and 1330 set up the string which follows the ampersand. The length will be in the A-register, and the address of the first character in INDEX ($5E,$5F). Lines 1340-1420 copy the string data into the monitor's buffer at $200. The characters are moved in backwards order, after first storing a carriage return at the end. So far the code is very similar to that of Bob Stitt.

Lines 1440-1590 are very similar to code found in the mini-assembler (at $F538-$F559 in the Integer BASIC ROMs). MON.GETNUM parses one hexadecimal number, if present, and then returns with a modified form of the first non-hex character in the A-register. Lines 1480-1520 search the monitor's command table for a matching character. If none is found, you will hear a bell. If found, the carriage return command is a special case. Lines 1540-1590 handle the carriage return command, and lines 1440-1450 handle all the others.

When the command has been fully parsed and executed, control will return to your Applesoft program. That is, unless your command had the effect of aborting Applesoft. Here is a sample program:

```
     10 PRINT CHR$(4)"BLOAD B.MONITOR" : CALL 768
     20 INPUT C$
     30 & C$
     40 PRINT : GOTO 20
```

Note that to type in a command which contains a ":" you will have to type a leading quotation mark. Otherwise Applesoft will issue its "EXTRA IGNORED" message and truncate your input at the colon.

Inside your Applesoft program you can build the command string using any sort of string functions and concatenation you wish.

```
                  1000 *SAVE S.AMPER MONITOR
                  1010 *-------------------------------
                  1020 *       &MONITOR
                  1030 *-------------------------------
03F5-             1040 AMPERSAND.VECTOR .EQ $3F5
DD7B-             1050 AS.FRMEVL   .EQ $DD7B
E5FD-             1060 AS.FRESTR   .EQ $E5FD
                  1070 *-------------------------------
0031-             1080 MON.MODE    .EQ $31
0034-             1090 MON.YSAV    .EQ $34
005E-             1100 INDEX       .EQ $5E,5F
                  1110 *-------------------------------
0200-             1120 BUFFER      .EQ $200
FFBE-             1130 MON.TOSUB   .EQ $FFBE
FFA7-             1140 MON.GETNUM  .EQ $FFA7
FFCC-             1150 MON.CHRTBL  .EQ $FFCC
FF3A-             1160 MON.BELL    .EQ $FF3A
FE00-             1170 MON.BL1     .EQ $FE00
FFC7-             1180 MON.ZMODE   .EQ $FFC7
                  1190 *-------------------------------
                  1200        .OR $300
                  1210 *-------------------------------
0300- A9 4C       1220 SETUP  LDA #$4C     JMP OPCODE
0302- 8D F5 03    1230        STA AMPERSAND.VECTOR
0305- A9 10       1240        LDA #FAKE.MONITOR
0307- 8D F6 03    1250        STA AMPERSAND.VECTOR+1
030A- A9 03       1260        LDA /FAKE.MONITOR
030C- 8D F7 03    1270        STA AMPERSAND.VECTOR+2
030F- 60          1280        RTS
                  1290 *-------------------------------
                  1300 FAKE.MONITOR
0310- 20 C7 FF    1310        JSR MON.ZMODE
0313- 20 7B DD    1320        JSR AS.FRMEVL
0316- 20 FD E5    1330        JSR AS.FRESTR
0319- A8          1340        TAY
031A- A9 8D       1350        LDA #$8D     <RETURN>
031C- 99 00 02    1360 .1     STA BUFFER,Y
031F- 98          1370        TYA
0320- F0 0C       1380        BEQ FMN2     END OF STRING
0322- 88          1390        DEY
0323- B1 5E       1400        LDA (INDEX),Y
0325- 09 80       1410        ORA #$80
0327- D0 F3       1420        BNE .1       ...ALWAYS
                  1430 *-------------------------------
0329- 20 BE FF    1440 FMN1   JSR MON.TOSUB
032C- A4 34       1450        LDY MON.YSAV
032E- 20 A7 FF    1460 FMN2   JSR MON.GETNUM
0331- 84 34       1470        STY MON.YSAV
0333- A0 16       1480        LDY #22
0335- D9 CC FF    1490 .1     CMP MON.CHRTBL,Y
0338- F0 06       1500        BEQ .2
033A- 88          1510        DEY
033B- 10 F8       1520        BPL .1
033D- 4C 3A FF    1530        JMP MON.BELL
0340- C0 15       1540 .2     CPY #21
0342- D0 E5       1550        BNE FMN1
0344- A5 31       1560        LDA MON.MODE
0346- A0 00       1570        LDY #0
0348- C6 34       1580        DEC MON.YSAV
034A- 4C 00 FE    1590        JMP MON.BL1
```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).